MIIA0106

Python and C Programming Language

อาจารย์ สุทิศ องอาจ

ภาคการศึกษา 2 ปีการศึกษา 2567

MIIA0106	Python and C Programming Language		SUN-เช้า-23/03/2568
A	MON/2	D604	[CPA/1,CPAI/1,EMAT/1]#ALL#
	MON/3	MII203	
	MON/3	MII202 A	
	MON/3	MII202 B	
B	SAT/2	MII207 B	[CPA+/1,EMAT+/1,EMA/2]#ALL#
	SUN/1	MII202 A	
	SUN/1	MII202 B	
EP	- /0	-	

คาบที่ 1: แนะนำภาษา C และฟังก์ชันเบื้องต้น (6 ชม.)

แหล่งค้นคว้าข้อมูลเพิ่มเติม

ตำรา:

- The C Programming Language (โดย Brian Kernighan และ Dennis Ritchie)
- C Primer Plus

เว็บไซต์:

- https://www.w3schools.com/c/index.php
- Tutorialspoint: https://www.tutorialspoint.com/cprogramming/
- Cprogramming.com: https://www.cprogramming.com/
- GeeksforGeeks: https://www.geeksforgeeks.org/c-programming-language/

YouTube:

- ค้นหาคำว่า "C programming tutorial"

1.1. บรรยาย (3 ชม.)

1.1.1.ความเป็นมาและความสำคัญของภาษา C

ภาษาซี (C) ได้รับการออกแบบและพัฒนาขึ้นโดย Dennis Ritchie เมื่อปี ค.ศ.1972 ห้องปฏิบัติ การเบลล์ (Bell Laboratories) โดยออกแบบเพื่อใช้งานบนระบบปฏิบัติการ UNIX บนเครื่องเมนเฟรม คอมพิวเตอร์ DEC PDP-11 ซึ่งภาษาซีได้พัฒนามาจากภาษาบี (B) ที่พัฒนา โดย Ken Thompson ภาษาบีถูกพัฒนาบนพื้นฐานของภาษาบีซีพีแอล (BCPL)

นาย Dennis Ritchie ผู้ออกแบบและพัฒนาภาษาซี (C)

ที่มา https://www.timetoast.com/timelines/1569368

ในเวลาต่อมา ภาษาซีได้รับความนิยมสูง สถาบัน ANSI (American National Standards Institute) ได้สร้างมาตรฐานภาษาซีขึ้นมา เพื่อรับรองให้เป็นสากล ภายใต้ชื่อว่า ANSI-C ตั้งแต่ปี ค.ศ.1983 และในปัจจุบันได้มี

การพัฒนาภาษาซีให้มีประสิทธิภาพมากยิ่งขึ้น เป็นเวอร์ชันต่าง ๆ มากมาย มีการพัฒนาต่อยอดเป็นภาษาซีพลัสพลัส (C++) หรือภาษาซีชาร์ป (C#) ซึ่งมีการเพิ่มชุดคำสั่งที่สนับสนุนการพัฒนาโปรแกรมเชิงวัตถุ (Object-Oriented Programming) และยังคงรองรับชุดคำสั่งมาตรฐานของภาษาซี คือ ANSI-C อยู่ด้วย

ภาษาซีเป็นโปรแกรมระดับสูง ที่ใช้สำหรับเขียนโปรแกรมประยุกต์ต่าง ๆ เช่นเดียวกันกับ ภาษาปาสคาล ภาษาเบสิก และภาษาฟอร์แทรน เป็นต้น นอกจากนี้ภาษาซียังใช้สำหรับเขียนโปรแกรมระบบ และโปรแกรมสำหรับควบคุมฮาร์ดแวร์บางส่วนที่โปรแกรมระดับสูงหลายภาษาไม่สามารถทำได้ (https://www.yupparaj.ac.th/thanphisit/bot1-1.html)

คุณสมบัติเด่น:

- ภาษาระดับกลาง: มีทั้งลักษณะของภาษาระดับสูง (อ่านเข้าใจง่าย) และภาษาระดับต่ำ (ควบคุมฮาร์ดแวร์ได้)
- มีประสิทธิภาพสูง: โปรแกรมที่เขียนด้วย C มักทำงานได้เร็วและใช้หน่วยความจำน้อย
- มีความยืดหยุ่น: สามารถนำไปพัฒนาโปรแกรมประเภทต่างๆ ได้อย่างหลากหลาย ทั้งระบบปฏิบัติการ, ฐานข้อมูล, คอมไพเลอร์, และแอปพลิเคชันทั่วไป
- ความสำคัญ: เป็นรากฐานของภาษาโปรแกรมสมัยใหม่หลายภาษา เช่น C++, Java, C# และเป็นภาษาที่นิยมใช้ในการพัฒนาระบบฝังตัว (embedded systems)

1.1.2.โครงสร้างพื้นฐานของโปรแกรมภาษา C

โครงสร้างทั่วไป:

```
#include <stdio.h>

int main() {
    // คำสั่งต่างๆ
    return 0;
}
```

ส่วนประกอบหลัก:

#include <stdio.h>: ประกาศใช้ไลบรารี stdio.h ซึ่งมีฟังก์ชันสำหรับการรับเข้าและแสดงผลข้อมูล

int main(): ฟังก์ชันหลักที่โปรแกรมเริ่มทำงาน

// คำสั่งต่างๆ: ส่วนที่บรรจุคำสั่งของโปรแกรม

ขั้นตอนที่ 1: การเขียนโปรแกรม (Source Code)

ในขั้นตอนแรกของการพัฒนาโปรแกรมด้วยภาษา C คุณจะต้องเขียนโปรแกรมโดยใช้ **Editor** และทำการบันทึกไฟล์ด้วยนามสกุล .c เช่น work.c เป็นต้น

Editor คืออะไร?

Editor คือโปรแกรมสำหรับการเขียนและแก้ไขโค้ดโปรแกรม มีหน้าที่ช่วยให้การเขียนโค้ดง่ายขึ้น โดยให้เครื่องมือสำหรับการจัดการข้อความและคำสั่งที่ใช้ในโปรแกรมได้อย่างสะดวก

ตัวอย่างของ **Editor** ที่นิยมใช้ในการเขียนโปรแกรม ได้แก่:

- **Notepad**: โปรแกรมแก้ไขข้อความพื้นฐานที่มาพร้อมกับระบบปฏิบัติการ Windows
- **TextPad**: Editor ที่มีความสามารถเพิ่มขึ้น เช่น การแสดงเลขบรรทัด และรองรับการเขียนโค้ดในหลายภาษา
- **EditPlus**: โปรแกรมแก้ไขโค้ดที่มีคุณสมบัติช่วยในการเขียนโปรแกรม เช่น การไฮไลต์โค้ดและคำสั่ง (Syntax Highlighting)
- **Vim/Emacs**: โปรแกรม Editor สำหรับนักพัฒนาที่เชี่ยวชาญ โดยนิยมใช้งานในระบบปฏิบัติการ Linux/Unix
- **Integrated Development Environment (IDE)**: ตัวอย่างเช่น Visual Studio Code (VS Code) หรือ Code::Blocks ซึ่งเป็น Editor ที่มาพร้อมเครื่องมือช่วยพัฒนาโปรแกรมภาษา C อย่างครบถ้วน

การบันทึกไฟล์โปรแกรม

หลังจากเขียนโปรแกรมเสร็จแล้ว คุณจะต้องบันทึกไฟล์โค้ดให้มีนามสกุล .c เช่น:

work.c
program1.c

การตั้งชื่อไฟล์ควรสะท้อนถึงหน้าที่หรือเนื้อหาของโปรแกรม เพื่อให้ง่ายต่อการจัดการและค้นหาในอนาคต

ขั้นตอนการเลือก Editor

การเลือก Editor ควรพิจารณาจาก:

- ความคุ้นเคย: เลือก Editor ที่คุณถนัดใช้งานมากที่สุด
- ฟีเจอร์ที่รองรับ: หากคุณต้องการเครื่องมือที่ช่วยตรวจสอบข้อผิดพลาดและจัดรูปแบบโค้ด ควรเลือก IDE เช่น Visual Studio Code
- ระบบปฏิบัติการ: ตรวจสอบว่า Editor ที่คุณเลือกสามารถทำงานได้ในระบบปฏิบัติการของคุณ (Windows, macOS, Linux)

- คำแนะนำเพิ่มเติมสำหรับมือใหม่ หากคุณเป็นมือใหม่ ขอแนะนำให้เริ่มต้นใช้ IDE เช่น Code::Blocks หรือ Visual Studio Code เพราะมีฟีเจอร์ช่วยเขียนโค้ด ลดความผิดพลาด และมีคู่มือการใช้งานที่ชัดเจน

ตัวอย่างโค้ดภาษา C เบื้องต้น

```
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

บันทึกไฟล์นี้เป็น hello.c และคอมไพล์โปรแกรมเพื่อทดสอบการทำงานของโค้ดดังกล่าว

ขั้นตอนที่ 2: การคอมไพล์โปรแกรม (Compile)

หลังจากเขียน Source Code ในขั้นตอนแรกเสร็จเรียบร้อยแล้ว ขั้นตอนต่อมาคือการ **คอมไพล์** ซึ่งเป็นการแปลง Source Code จากภาษาที่มนุษย์เข้าใจ (ภาษา C) ไปเป็นภาษาที่เครื่องคอมพิวเตอร์เข้าใจ (ภาษาเครื่อง)

การทำงานของคอมไพเลอร์ (Compiler)

คอมไพเลอร์จะทำการ:

1. **ตรวจสอบข้อผิดพลาดในโค้ด (Error Checking)**
   - คอมไพเลอร์จะตรวจสอบ Source Code เพื่อดูว่ามีข้อผิดพลาดทางไวยากรณ์หรือไม่ เช่น การลืมใส่ `;` หรือการเรียกใช้ฟังก์ชันที่ไม่มีการประกาศล่วงหน้า
   - กรณีที่พบข้อผิดพลาด คอมไพเลอร์จะแสดงข้อความแจ้งข้อผิดพลาดพร้อมระบุบรรทัดที่ผิดพลาด ผู้เขียนโปรแกรมจะต้องกลับไปแก้ไขโค้ดและทำการคอมไพล์ใหม่
   - กรณีที่ไม่พบข้อผิดพลาด คอมไพเลอร์จะแปล Source Code เป็นไฟล์ภาษาเครื่องที่มีนามสกุล `.obj` เช่น ไฟล์ `work.c` จะถูกแปลเป็น `work.obj`

2. การแปลงเป็นภาษาเครื่อง

คอมไพเลอร์จะแปลงโปรแกรมทั้งหมดในขั้นตอนเดียว โดยอ่าน Source Code ตั้งแต่ต้นจนจบ และสร้างไฟล์ภาษาเครื่อง (Object File) ซึ่งคอมพิวเตอร์สามารถเข้าใจและนำไปทำงานต่อได้

3. ตัวอย่างผลลัพธ์จากการคอมไพล์

- Source Code: `work.c`
- ผลลัพธ์ภาษาเครื่อง: `work.obj`

คำอธิบายเพิ่มเติมเกี่ยวกับคอมไพเลอร์และอินเตอร์พรีเตอร์

- คอมไพเลอร์ (Compiler)

คอมไพเลอร์เป็นโปรแกรมแปลภาษารูปแบบหนึ่ง โดยจะแปลงโค้ดทั้งหมดในคราวเดียว (Batch Translation) หลังจากแปลงเสร็จ ไฟล์ที่ได้จะสามารถรันได้โดยตรงโดยไม่ต้องพึ่ง Source Code

- อินเตอร์พรีเตอร์ (Interpreter)

อินเตอร์พรีเตอร์จะแปลงและรันโค้ดทีละบรรทัด กล่าวคือ:

1. อ่านคำสั่งบรรทัดแรก
2. แปลคำสั่งบรรทัดแรกเป็นภาษาเครื่อง
3. ทำงานตามคำสั่งในบรรทัดนั้น
4. ทำซ้ำขั้นตอนข้างต้นไปจนจบโปรแกรม

เปรียบเทียบ: คอมไพเลอร์ vs อินเตอร์พรีเตอร์

คุณสมบัติ	**คอมไพเลอร์**	**อินเตอร์พรีเตอร์**
วิธีการแปล	แปลทั้งหมดในครั้งเดียว	แปลทีละบรรทัด
ความเร็วในการรันโปรแกรม	เร็วกว่า เพราะแปลทั้งหมดล่วงหน้าแล้ว	ช้ากว่า เพราะต้องแปลทีละบรรทัดในขณะรัน
ผลลัพธ์การแปล	ได้ไฟล์ภาษาเครื่องที่พร้อมรันได้ทันที	ต้องมี Source Code เพื่อรันโปรแกรม
การตรวจสอบข้อผิดพลาด	แจ้งข้อผิดพลาดทั้งหมดในครั้งเดียว	แจ้งข้อผิดพลาดทีละบรรทัด

ภาษาโปรแกรมที่ใช้คอมไพเลอร์ (Compiler)

คอมไพเลอร์จะแปลงโค้ดทั้งหมดในครั้งเดียว และสร้างไฟล์ภาษาเครื่องที่สามารถรันได้โดยตรง

ภาษาโปรแกรม	คอมไพเลอร์ที่นิยมใช้
C	GCC, Clang, MSVC
C++	GCC, Clang, MSVC
Java (แปลงเป็น bytecode)	javac (Java Compiler)
Go	Go Compiler (GCC Go)
Rust	Rust Compiler (rustc)
Fortran	GNU Fortran (gfortran), Intel Fortran
Pascal	Free Pascal, Turbo Pascal

ภาษาโปรแกรมที่ใช้อินเตอร์พรีเตอร์ (Interpreter)

อินเตอร์พรีเตอร์จะแปลงและรันโค้ดทีละบรรทัดในขณะทำงาน

ภาษาโปรแกรม	**อินเตอร์พรีเตอร์ที่นิยมใช้**
Python	CPython, PyPy, Jython
JavaScript	V8 (ใช้ใน Chrome), SpiderMonkey (ใช้ใน Firefox)
Ruby	MRI (Matz's Ruby Interpreter), JRuby
PHP	PHP Interpreter
Perl	Perl Interpreter
Bash (Shell Script)	Bash Interpreter
MATLAB	MATLAB Interpreter

ภาษาโปรแกรมที่รองรับทั้งคอมไพเลอร์และอินเตอร์พรีเตอร์

บางภาษาออกแบบให้รองรับทั้งสองแบบ ขึ้นอยู่กับสภาพแวดล้อมที่ใช้งาน เช่น:

Java

ใช้คอมไพเลอร์ (javac) เพื่อแปลง Source Code เป็น bytecode

ใช้ JVM (Java Virtual Machine) เพื่อรัน bytecode ด้วยการอินเตอร์เพรตหรือ Just-In-Time Compilation (JIT)

Python

ปกติ Python ใช้อินเตอร์พรีเตอร์ (เช่น CPython)

สามารถคอมไพล์เป็น bytecode (ไฟล์ .pyc) เพื่อเร่งความเร็วในการรัน

JavaScript

โดยทั่วไปใช้อินเตอร์พรีเตอร์ในเบราว์เซอร์ (เช่น V8 Engine)

แต่บางแพลตฟอร์มใช้การคอมไพล์ล่วงหน้า (Ahead-of-Time Compilation) เพื่อประสิทธิภาพ

สรุป

คอมไพเลอร์: นิยมใช้ในภาษาโปรแกรมที่ต้องการประสิทธิภาพสูง เช่น C, C++, Rust

อินเตอร์พรีเตอร์: นิยมใช้ในภาษาโปรแกรมที่เน้นความยืดหยุ่นและง่ายต่อการพัฒนา เช่น Python, JavaScript

บางภาษาอย่าง Java และ Python รองรับทั้งสองแบบขึ้นอยู่กับความต้องการของผู้พัฒนา

แนวคิดในการเขียนโปรแกรมคอมพิวเตอร์

การพัฒนาโปรแกรมคอมพิวเตอร์มีขั้นตอนสำคัญ 5 ขั้นตอน ซึ่งแต่ละขั้นตอนมีบทบาทสำคัญในการทำให้โปรแกรมสามารถแก้ไขปัญหาได้อย่างมีประสิทธิภาพและตรงตามความต้องการของผู้ใช้งาน

ตัวอย่างโจทย์: การพัฒนาโปรแกรมคำนวณเกรดนักเรียน

โจทย์: สร้างโปรแกรมที่รับคะแนนนักเรียน และคำนวณเกรดตามเกณฑ์ที่กำหนด เช่น

- คะแนน 80-100: เกรด A
- คะแนน 70-79: เกรด B
- คะแนน 60-69: เกรด C
- คะแนน 50-59: เกรด D
- คะแนนน้อยกว่า 50: เกรด F

1. วิเคราะห์ปัญหา (Analysis)

ในขั้นตอนนี้ จะต้องทำความเข้าใจปัญหาให้ชัดเจนโดยตอบคำถาม เช่น:

โจทย์ต้องการอะไร?

ต้องการโปรแกรมที่สามารถคำนวณเกรดตามคะแนนที่ป้อนเข้ามา

ข้อมูลที่ต้องใช้มีอะไรบ้าง?

- Input: คะแนนนักเรียน (ตัวเลข 0-100)
- Output: เกรด (A, B, C, D, F)

กฎเกณฑ์หรือเงื่อนไขที่ต้องพิจารณา?

เกณฑ์คะแนนสำหรับการตัดเกรด

2. วางแผนและออกแบบ (Planning & Design)

ในขั้นตอนนี้จะออกแบบโครงสร้างโปรแกรม วิธีการทำงาน และหน้าตาของโปรแกรม เช่น:

ร่างขั้นตอนการทำงาน (Algorithm):

- รับคะแนนจากผู้ใช้งาน
- ตรวจสอบว่าอยู่ในช่วง 0-100 หรือไม่ (ป้องกันการป้อนค่าผิดพลาด)
- เปรียบเทียบคะแนนกับเกณฑ์ที่กำหนดเพื่อกำหนดเกรด
- แสดงผลลัพธ์ (เกรด)

ผังงาน (Flowchart):

- ใช้สัญลักษณ์และลูกศรเพื่อแสดงลำดับขั้นตอนการทำงาน

ซูโดโค้ด (Pseudo Code)

ซูโดโค้ดเป็นเครื่องมือช่วยอธิบายขั้นตอนการทำงานของโปรแกรมในรูปแบบที่เข้าใจง่าย โดยไม่ขึ้นอยู่กับภาษาโปรแกรมใด ๆ ซูโดโค้ดจะใช้โครงสร้างคล้ายภาษาโปรแกรม แต่เขียนด้วยภาษาที่มนุษย์อ่านเข้าใจง่าย เช่น ภาษาไทยหรือภาษาอังกฤษ

คุณสมบัติของซูโดโค้ด

1. ไม่ขึ้นกับภาษาโปรแกรม
2. เน้นโครงสร้างการทำงานของโปรแกรม
3. ใช้สำหรับวางแผนหรือออกแบบขั้นตอนวิธี (Algorithm)
4. อ่านง่ายสำหรับทั้งนักพัฒนาและผู้ไม่มีพื้นฐานการเขียนโปรแกรม

การออกแบบอินเทอร์เฟซ:

- หากเป็นโปรแกรมแบบข้อความ ออกแบบให้ผู้ใช้ป้อนคะแนนในรูปแบบที่ชัดเจน
- หากเป็นโปรแกรมแบบกราฟิก ออกแบบให้ใช้งานง่าย เช่น กล่องป้อนข้อมูลและปุ่มคำนวณ

3. เขียนโปรแกรม (Coding)

ในขั้นตอนนี้ ใช้ภาษาโปรแกรม เช่น C, Python หรือ JavaScript เขียนโค้ดตามแผนที่ออกแบบไว้

ตัวอย่างโค้ด (Python):

```
# รับคะแนนจากผู้ใช้
score = float(input("Enter the student's score (0-100): "))

# ตรวจสอบคะแนนและกำหนดเกรด
if 80 <= score <= 100:
    grade = 'A'
elif 70 <= score < 80:
    grade = 'B'
elif 60 <= score < 70:
    grade = 'C'
elif 50 <= score < 60:
    grade = 'D'
elif 0 <= score < 50:
    grade = 'F'
else:
    grade = 'Invalid score'  # สำหรับค่าที่อยู่นอกช่วง

# แสดงผลลัพธ์
print(f"The grade is: {grade}")
```

4. ทดสอบโปรแกรม (Testing)

หลังจากเขียนโค้ดเสร็จ จะต้องทดสอบเพื่อให้แน่ใจว่าโปรแกรมทำงานถูกต้อง:

ทดสอบตามกรณีที่กำหนด:

- กรณีปกติ: คะแนนในช่วง 0-100 เช่น 85 (ต้องได้เกรด A)
- กรณีพิเศษ: คะแนนที่อยู่นอกช่วง เช่น -10 หรือ 150 (ต้องแสดง "Invalid score")

การทดสอบแบบ Edge Cases:

- คะแนนที่ใกล้กับขอบช่วง เช่น 79, 80, 49, 50 เพื่อดูว่าโปรแกรมจัดเกรดได้ถูกต้องหรือไม่

แก้ไขข้อผิดพลาด:

- หากพบว่าโปรแกรมมีข้อผิดพลาด (Bug) ให้ทำการปรับปรุงโค้ดและทดสอบใหม่

5. จัดทำคู่มือ (Documentation)

การจัดทำเอกสารหรือคู่มือช่วยให้ผู้ใช้งานและนักพัฒนาคนอื่นเข้าใจการทำงานของโปรแกรม:

คู่มือสำหรับผู้ใช้งาน (User Manual):

- อธิบายวิธีการใช้งาน เช่น การป้อนคะแนน และการอ่านผลลัพธ์

คู่มือสำหรับนักพัฒนา (Developer Documentation):

- อธิบายโครงสร้างโค้ด วิธีการทำงานของแต่ละส่วน
- อาจรวมถึง Flowchart หรือ Pseudocode เพื่อช่วยในการแก้ไขโปรแกรมในอนาคต

โครงสร้างของภาษา C/C++

ภาษา C และ C++ มีโครงสร้างพื้นฐานที่คล้ายคลึงกัน โดยสามารถแบ่งออกเป็น 6 ส่วนหลัก ดังนี้:

1. **การนำเข้าหัวข้อไลบรารี:** ใช้สำหรับเรียกใช้ฟังก์ชันหรือคลาสสำเร็จรูป
2. **การประกาศข้อมูล:** กำหนดตัวแปรหรือค่าที่ใช้ทั้งโปรแกรม
3. **ฟังก์ชันหลัก:** จุดเริ่มต้นการทำงานของโปรแกรม
4. **ฟังก์ชันอื่น:** สำหรับการแบ่งงานหรือสร้างฟังก์ชันเฉพาะ
5. **ตัวแปรและนิพจน์:** ใช้สำหรับเก็บและประมวลผลข้อมูล
6. **คำอธิบาย:** ช่วยให้โค้ดเข้าใจง่ายขึ้น

โครงสร้างนี้ช่วยให้โปรแกรมมีความชัดเจนและง่ายต่อการพัฒนา

1. ส่วนการนำเข้าหัวข้อไลบรารี (Preprocessor Directives)

- ส่วนนี้ใช้สำหรับการนำเข้าหรือรวมไฟล์ไลบรารีที่จำเป็น เช่น ฟังก์ชันพื้นฐานหรือฟังก์ชันเฉพาะที่ต้องการใช้ในโปรแกรม
- ในภาษา C/C++ ใช้คำสั่ง #include เพื่อเรียกใช้ไลบรารี

```
#include <iostream>  // ใช้สำหรับการแสดงผลในภาษา C++
#include <stdio.h>   // ใช้สำหรับการแสดงผลในภาษา C
```

2. ส่วนการประกาศข้อมูล (Global Declaration)

- ใช้สำหรับประกาศตัวแปรหรือฟังก์ชันที่สามารถเข้าถึงได้จากทุกส่วนของโปรแกรม (Global Scope)
- ตัวแปรที่ประกาศในส่วนนี้จะมีผลกับทุกฟังก์ชันในโปรแกรม

```
int globalVar = 10;  // ตัวแปร global
```

3. ส่วนฟังก์ชันหลัก (Main Function)

- เป็นส่วนสำคัญของโปรแกรม เพราะโปรแกรมจะเริ่มทำงานที่ฟังก์ชัน main()
- ทุกโปรแกรมใน C/C++ จะต้องมีฟังก์ชันนี้

```
int main() {
    std::cout << "Hello, World!" << std::endl;  // ภาษา C++
    return 0;  // แสดงว่าการทำงานเสร็จสมบูรณ์
}
```

4. ส่วนฟังก์ชันอื่น (User-defined Functions)

- ใช้สำหรับสร้างฟังก์ชันที่ผู้เขียนโปรแกรมกำหนดเอง เพื่อแบ่งงานและลดความซับซ้อนของโค้ด
- ฟังก์ชันเหล่านี้สามารถเรียกใช้ใน main() หรือฟังก์ชันอื่น ๆ ได้

```
int addNumbers(int a, int b) {
    return a + b;  // ฟังก์ชันบวกตัวเลขสองตัว
}
```

5. ส่วนตัวแปรและนิพจน์ (Variables and Statements)

- ใช้สำหรับประกาศตัวแปรภายในฟังก์ชันและเขียนนิพจน์ (Expressions) หรือคำสั่ง (Statements)
- ตัวแปรเหล่านี้มีขอบเขตการใช้งาน (Scope) ภายในฟังก์ชันหรือบล็อกที่ประกาศ

```
int main() {
    int num1 = 5, num2 = 10;  // ประกาศตัวแปรภายในฟังก์ชัน main
    int sum = num1 + num2;    // นิพจน์การบวก
    std::cout << "Sum: " << sum << std::endl;
    return 0;
}
```

6. ส่วนคำอธิบายโปรแกรม (Comments)

- ใช้สำหรับใส่คำอธิบายหรือหมายเหตุในโปรแกรม เพื่อช่วยให้อ่านโค้ดง่ายขึ้น
- คำอธิบายไม่มีผลต่อการทำงานของโปรแกรม

```
// นี่คือคำอธิบายแบบบรรทัดเดียว
/* นี่คือคำอธิบายแบบหลายบรรทัด */
```

```
#include <iostream>  // ส่วนการนำเข้าหัวข้อไลบรารี

int globalVar = 10;  // ส่วนการประกาศข้อมูล (Global Declaration)

// ส่วนฟังก์ชันที่ผู้ใช้กำหนดเอง
int addNumbers(int a, int b) {
    return a + b;  // ฟังก์ชันบวกตัวเลขสองตัว
}

// ฟังก์ชันหลักของโปรแกรม
int main() {
    // ประกาศตัวแปร
    int num1 = 20, num2 = 30;

    // เรียกใช้ฟังก์ชัน addNumbers
    int sum = addNumbers(num1, num2);

    // แสดงผลลัพธ์
    std::cout << "Sum: " << sum << std::endl;

    return 0;  // จบการทำงานของโปรแกรม
}
```

1.1.3.ตัวแปร ชนิดข้อมูล และค่าคงที่ใน C และ C++

ในภาษา C และ C++ การเขียนโปรแกรมเกี่ยวข้องกับการเก็บและประมวลผลข้อมูล ซึ่งต้องเข้าใจ **ตัวแปร (Variables)**, **ชนิดข้อมูล (Data Types)**, และ **ค่าคงที่ (Constants)** อย่างชัดเจน

1. ตัวแปร (Variables)

ตัวแปร คือชื่อที่ใช้แทนตำแหน่งหน่วยความจำที่เก็บข้อมูล ตัวแปรแต่ละตัวจะมีชื่อ ชนิดข้อมูล และค่า

การประกาศตัวแปร

รูปแบบทั่วไป:

```
data_type variable_name;
```

เช่น

```
int age;  // ตัวแปรเก็บจำนวนเต็ม
float score;  // ตัวแปรเก็บจำนวนทศนิยม
```

ตัวอย่างโค้ด: การประกาศและใช้งานตัวแปร

```
#include <iostream>
using namespace std;

int main() {
    int age = 25;              // ประกาศตัวแปรจำนวนเต็ม
    float score = 85.5;        // ประกาศตัวแปรทศนิยม
    char grade = 'A';          // ประกาศตัวแปรตัวอักษร
    bool isPass = true;        // ประกาศตัวแปรค่าความจริง

    cout << "Age: " << age << endl;
    cout << "Score: " << score << endl;
    cout << "Grade: " << grade << endl;
    cout << "Pass: " << (isPass ? "Yes" : "No") << endl;

    return 0;
}
```

ผลลัพธ์ที่ได้:

```
Age: 25
Score: 85.5
Grade: A
Pass: Yes
```

2. ชนิดข้อมูลพื้นฐาน (Basic Data Types)

ชนิดข้อมูล	คำอธิบาย	ตัวอย่าง
int	เก็บจำนวนเต็ม (บวก, ลบ, หรือศูนย์)	int age = 30;
float	เก็บจำนวนทศนิยม (32 บิต)	float score = 85.75;
double	เก็บจำนวนทศนิยมที่แม่นยำกว่า (64 บิต)	double distance = 1000.12345;
char	เก็บตัวอักษร 1 ตัว	char grade = 'A';
bool	เก็บค่าความจริง (true หรือ false)	bool isPass = true;

หมายเหตุ:

- ขนาดของชนิดข้อมูลขึ้นอยู่กับระบบ (ส่วนใหญ่: int = 4 bytes, float = 4 bytes, double = 8 bytes)

ชนิดข้อมูลเพิ่มเติมใน C++

C++ มีชนิดข้อมูลเพิ่มเติม เช่น string สำหรับเก็บข้อความ:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name = "John Doe";  // ข้อความ
    cout << "Name: " << name << endl;
    return 0;
}
```

3. ค่าคงที่ (Constants)

ค่าคงที่ คือค่าที่ไม่สามารถเปลี่ยนแปลงได้หลังจากกำหนดแล้ว

- ใช้สำหรับค่าที่ไม่ต้องการให้มีการเปลี่ยนแปลงในโปรแกรม เช่น ค่า PI, หรือค่าที่เป็นการตั้งค่าตายตัว

วิธีการประกาศค่าคงที่

- **ใช้ const keyword**:

const data_type constant_name = value;

ตัวอย่าง

```
const double PI = 3.14159;
const int MAX_SCORE = 100;
```

- **ใช้ #define (C/C++ Preprocessor)**:

#define constant_name value

ตัวอย่าง

```
#define PI 3.14159
#define MAX_SCORE 100
```

ตัวอย่างโค้ด: ใช้ค่าคงที่

```
#include <iostream>
#define PI 3.14159  // ค่าคงที่แบบสัญลักษณ์

int main() {
    const int MAX_SCORE = 100;  // ค่าคงที่แบบ const
    double radius = 10.0;
    double area = PI * radius * radius;  // คำนวณพื้นที่วงกลม
    std::cout << "Max Score: " << MAX_SCORE << std::endl;
    std::cout << "Circle Area: " << area << std::endl;
    // MAX_SCORE = 90;  // Error: ไม่สามารถเปลี่ยนค่าคงที่
    return 0;
}
```

ผลลัพธ์ที่ได้:

```
Max Score: 100
Circle Area: 314.159
```

สรุปความแตกต่างระหว่างตัวแปรและค่าคงที่

คุณลักษณะ	ตัวแปร	ค่าคงที่
การเปลี่ยนแปลง	เปลี่ยนค่าได้ระหว่างการทำงาน	ไม่สามารถเปลี่ยนค่าได้หลังจากกำหนด
คำสำคัญที่ใช้	ไม่ต้องใช้คำสำคัญเพิ่มเติม	ใช้ const หรือ #define
ตัวอย่าง	int age = 20;	const int age = 20;

หลักการตั้งชื่อตัวแปรใน C และ C++

1. หลักการตั้งชื่อตัวแปร

- **ต้องขึ้นต้นด้วยตัวอักษรหรือเครื่องหมายขีดล่าง (_)**
  ห้ามขึ้นต้นด้วยตัวเลข
- **ใช้ตัวอักษรภาษาอังกฤษ ตัวเลข และ _ เท่านั้น**
  ห้ามใช้สัญลักษณ์พิเศษ เช่น @, #, $
- **ต้องไม่ตรงกับคำสงวน (Reserved Keywords)**
  เช่น int, return, class
- **ตั้งชื่อให้สื่อความหมาย**
  ควรตั้งชื่อที่อธิบายหน้าที่หรือข้อมูลที่เก็บ
- **ความยาวไม่ควรเกิน 255 ตัวอักษร**
  แต่ควรสั้นและกระชับ
- **Case-sensitive**
  ชื่อตัวแปรใน C และ C++ แยกแยะตัวพิมพ์เล็กและตัวพิมพ์ใหญ่ เช่น Age และ age ถือว่าเป็นคนละตัวแปร

2. ตัวอย่างการตั้งชื่อตัวแปร

ตัวแปรที่ถูกต้อง	**คำอธิบาย**
age	เก็บอายุ
_score	ใช้เครื่องหมาย _ ขึ้นต้น
studentName	เก็บชื่อนักเรียน
total_price	เก็บราคาสินค้ารวม
ตัวแปรที่ไม่ถูกต้อง	**เหตุผลที่ผิด**
2ndScore	ห้ามขึ้นต้นด้วยตัวเลข
total-price	ห้ามใช้เครื่องหมาย -
class	ชื่อตรงกับคำสงวน
@name	ห้ามใช้สัญลักษณ์พิเศษ

3. การตั้งชื่อให้ตรงกับข้อมูล

- ตั้งชื่อตัวแปรให้บ่งบอกถึงข้อมูลที่เก็บเพื่อให้อ่านโค้ดง่ายและเข้าใจหน้าที่ของตัวแปร
- ใช้ **Camel Case** หรือ **Snake Case** เป็นมาตรฐาน
    - **Camel Case:** ขึ้นต้นด้วยตัวพิมพ์เล็ก และตัวอักษรแรกของคำถัดไปเป็นตัวพิมพ์ใหญ่ เช่น studentName, totalPrice
    - **Snake Case:** ใช้ _ เชื่อมคำ เช่น student_name, total_price

ตัวอย่างการตั้งชื่อให้ตรงกับข้อมูล:

ข้อมูล	**ชื่อตัวแปรที่เหมาะสม**
อายุ	age
คะแนน	score
ชื่อนักเรียน	studentName หรือ student_name
จำนวนสินค้า	itemCount หรือ item_count
ราคาสินค้ารวม	totalPrice หรือ total_price

4. ชื่อตัวแปรที่สงวนไว้ (Reserved Keywords)

C และ C++ มีคำสงวนที่ไม่สามารถใช้เป็นชื่อตัวแปร เพราะเป็นคำที่มีความหมายเฉพาะในภาษา เช่น:

- **คำสงวนใน C:**

  auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while
- **คำสงวนเพิ่มเติมใน C++:**

  asm, bool, catch, class, const_cast, delete, dynamic_cast, explicit, false, friend, inline, mutable, namespace, new, operator, private, protected, public, reinterpret_cast, static_cast, template, this, throw, true, try, typeid, typename, using, virtual, wchar_t

ตัวอย่างคำสงวน:

```
// ตัวอย่างคำสงวนที่ผิด
int class;     // Error: class เป็นคำสงวน
float return; // Error: return เป็นคำสงวน


// วิธีแก้
int myClass;  // ชื่อที่ถูกต้อง
float result; // ชื่อที่เหมาะสม
```

ตัวอย่างโค้ดแสดงการตั้งชื่อตัวแปร

```
#include <iostream>
using namespace std;

int main() {
    // ตัวแปรที่สื่อความหมาย
    int age = 20;                   // อายุ
    float totalPrice = 450.75;      // ราคาสินค้ารวม
    string studentName = "John";   // ชื่อนักเรียน
    bool isPass = true;             // ผ่านการสอบหรือไม่

    // แสดงค่าของตัวแปร
    cout << "Student Name: " << studentName << endl;
    cout << "Age: " << age << endl;
    cout << "Total Price: " << totalPrice << endl;
    cout << "Pass: " << (isPass ? "Yes" : "No") << endl;

    return 0;
}
```

ผลลัพธ์ที่ได้:

```
Student Name: John
Age: 20
Total Price: 450.75
Pass: Yes
```

สรุปหลักการตั้งชื่อตัวแปร

1. ห้ามขึ้นต้นด้วยตัวเลข
2. ห้ามใช้คำสงวน
3. ใช้ตัวอักษรภาษาอังกฤษ ตัวเลข และ _ เท่านั้น
4. ตั้งชื่อให้สื่อความหมาย
5. ใช้มาตรฐานการตั้งชื่อ เช่น **Camel Case** หรือ **Snake Case**

การตั้งชื่อตัวแปรที่ดีช่วยให้โค้ดอ่านง่ายและลดข้อผิดพลาดในการพัฒนาโปรแกรม

1.1.1. ตัวดำเนินการทางคณิตศาสตร์และตรรกะใน C และ C++

ตัวดำเนินการใน C และ C++ แบ่งออกเป็น **ตัวดำเนินการทางคณิตศาสตร์** และ **ตัวดำเนินการทางตรรกะ** โดยใช้สำหรับการคำนวณและการเปรียบเทียบค่าในโปรแกรม

1. ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

ใช้สำหรับการคำนวณทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร และหาค่าหรือเศษ

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง
+	บวก	a + b
-	ลบ	a - b
*	คูณ	a * b
/	หาร	a / b (หารจำนวนเต็มหรือทศนิยม)
%	หารเอาเศษ (Modulo)	a % b (ใช้ได้เฉพาะจำนวนเต็ม)

ตัวอย่างโค้ด: การใช้ตัวดำเนินการทางคณิตศาสตร์

```
#include <iostream>

using namespace std;

int main() {

    int a = 10, b = 3;

    cout << "a + b = " << (a + b) << endl;  // บวก

    cout << "a - b = " << (a - b) << endl;  // ลบ

    cout << "a * b = " << (a * b) << endl;  // คูณ

    cout << "a / b = " << (a / b) << endl;  // หารจำนวนเต็ม

    cout << "a % b = " << (a % b) << endl;  // หารเอาเศษ

    return 0;

}
```

ผลลัพธ์:

```
a + b = 13

a - b = 7

a * b = 30

a / b = 3

a % b = 1
```

2. ตัวดำเนินการทางตรรกะ (Logical Operators)

ใช้สำหรับการเปรียบเทียบค่าและการดำเนินการเชิงตรรกะ เช่น AND, OR, NOT

ตัวดำเนินการ	คำอธิบาย	ตัวอย่าง
&&	AND (และ)	(a > 0 && b < 5)
`		`
!	NOT (ไม่)	!(a > 0)

ตัวอย่างโค้ด: การใช้ตัวดำเนินการทางตรรกะ

```
#include <iostream>

using namespace std

int main() {

    int a = 5, b = 10;

    cout << "a > 0 AND b > 0: " << (a > 0 && b > 0) << endl;  // AND

    cout << "a > 0 OR b < 0: " << (a > 0 || b < 0) << endl;   // OR

    cout << "NOT (a > 0): " << !(a > 0) << endl;              // NOT

    return 0;

}
```

ผลลัพธ์:

```
a > 0 AND b > 0: 1

a > 0 OR b < 0: 1

NOT (a > 0): 0
```

3. ตัวดำเนินการเปรียบเทียบ (Relational Operators)

ใช้สำหรับการเปรียบเทียบค่าระหว่างตัวแปรหรือค่าคงที่

ตัว ดำเนินการ	คำอธิบาย	ตัวอย่าง
==	เท่ากับ	a == b
!=	ไม่เท่ากับ	a != b
>	มากกว่า	a > b
<	น้อยกว่า	a < b
>=	มากกว่าหรือ เท่ากับ	a >= b
<=	น้อยกว่าหรือ เท่ากับ	a <= b

ตัวอย่างโค้ด: การใช้ตัวดำเนินการเปรียบเทียบ

```
#include <iostream>

using namespace std;

int main() {

    int a = 5, b = 10;

    cout << "a == b: " << (a == b) << endl;  // เท่ากับ

    cout << "a != b: " << (a != b) << endl;  // ไม่เท่ากับ

    cout << "a > b: " << (a > b) << endl;    // มากกว่า

    cout << "a < b: " << (a < b) << endl;    // น้อยกว่า
```

```
    cout << "a >= b: " << (a >= b) << endl;  // มากกว่าหรือเท่ากับ
    cout << "a <= b: " << (a <= b) << endl;  // น้อยกว่าหรือเท่ากับ


    return 0;
}
```

ผลลัพธ์:

```
a == b: 0
a != b: 1
a > b: 0
a < b: 1
a >= b: 0
a <= b: 1
```

4. การผสมตัวดำเนินการทางคณิตศาสตร์และตรรกะ

สามารถใช้ตัวดำเนินการทั้งสองประเภทในเงื่อนไขร่วมกันได้ เช่น:

```
#include <iostream>
using namespace std;
int main() {
    int x = 15, y = 20, z = 25;
    if ((x < y) && (z > y)) {  // ใช้ทั้งเปรียบเทียบและ AND
        cout << "x น้อยกว่า y และ z มากกว่า y" << endl;
    }
    if ((x + y) > z || z < 30) {  // ใช้ทั้งคณิตศาสตร์และ OR
```

```
        cout << "ผลรวมของ x กับ y มากกว่า z หรือ z น้อยกว่า 30" << endl;
    }


    return 0;
}
```

ผลลัพธ์:

```
x น้อยกว่า y และ z มากกว่า y
ผลรวมของ x กับ y มากกว่า z หรือ z น้อยกว่า 30
```

สรุปการใช้ตัวดำเนินการใน C และ C++

1. **ตัวดำเนินการทางคณิตศาสตร์**: ใช้สำหรับคำนวณค่าต่าง ๆ เช่น บวก ลบ คูณ หาร
2. **ตัวดำเนินการทางตรรกะ**: ใช้สำหรับสร้างเงื่อนไขและการตัดสินใจ เช่น AND, OR, NOT
3. **ตัวดำเนินการเปรียบเทียบ**: ใช้สำหรับเปรียบเทียบค่าระหว่างตัวแปร เช่น เท่ากัน มากกว่า น้อยกว่า
4. สามารถผสมตัวดำเนินการต่าง ๆ เพื่อสร้างเงื่อนไขซับซ้อนได้

การเข้าใจและใช้ตัวดำเนินการเหล่านี้ได้อย่างถูกต้อง จะช่วยให้การเขียนโปรแกรมมีประสิทธิภาพและลดข้อผิดพลาด

1.1.2.คำสั่ง Input/Output พื้นฐานใน C และ C++

คำสั่ง Input/Output พื้นฐานใน C และ C++

การรับค่า (Input) และแสดงผล (Output) เป็นพื้นฐานสำคัญในโปรแกรมทุกภาษา สำหรับ **C** และ **C++** มีวิธีการดังนี้:

คำสั่ง Input/Output ในภาษา C

ในภาษา C ใช้ฟังก์ชันจากไลบรารี <stdio.h> เพื่อจัดการการรับค่าและแสดงผล

1. คำสั่งแสดงผล (Output)

ใช้ฟังก์ชัน printf() เพื่อแสดงข้อความหรือค่าของตัวแปร

```
printf("ข้อความที่ต้องการแสดง");
```

สามารถใช้ **Placeholder** เพื่อแสดงค่าตัวแปร เช่น %d, %f, %c

ตัวอย่าง:

```
#include <stdio.h>


int main() {
    int age = 20;
    printf("Hello, World!\n");
    printf("Age: %d\n", age);  // แสดงค่าตัวแปร age


    return 0;
}
```

ผลลัพธ์:

```
Hello, World!
Age: 20
```

2. คำสั่งรับค่า (Input)

ใช้ฟังก์ชัน scanf() เพื่อรับค่าจากผู้ใช้งาน

```
scanf("รูปแบบการรับข้อมูล", &ตัวแปร);
```

ต้องใช้เครื่องหมาย & (Address-of Operator) หน้าชื่อตัวแปร ยกเว้นกับ char[]

```
#include <stdio.h>



int main() {

    int age;

    printf("Enter your age: ");

    scanf("%d", &age);  // รับค่าจำนวนเต็มจากผู้ใช้

    printf("Your age is %d\n", age);



    return 0;

}
```

ผลลัพธ์:

```
Enter your age: 25

Your age is 25
```

คำสั่ง Input/Output ในภาษา C++

ภาษา C++ ใช้ **ไลบรารี <iostream>** และใช้คำสั่ง cin สำหรับ Input และ cout สำหรับ Output

1. คำสั่งแสดงผล (Output)

ใช้คำสั่ง cout พร้อมตัวดำเนินการ << เพื่อแสดงข้อความหรือค่าของตัวแปร

```
cout << "ข้อความที่ต้องการแสดง";
```

ตัวอย่าง:

```
#include <iostream>

using namespace std;



int main() {

    int age = 20;

    cout << "Hello, World!" << endl;  // endl ใช้ขึ้นบรรทัดใหม่

    cout << "Age: " << age << endl;   // แสดงค่าตัวแปร age



    return 0;

}
```

ผลลัพธ์:

```
Hello, World!

Age: 20
```

2. คำสั่งรับค่า (Input)

ใช้คำสั่ง cin พร้อมตัวดำเนินการ >> เพื่อรับค่าจากผู้ใช้งาน

```
cin >> ตัวแปร;
```

ตัวอย่าง

```
#include <iostream>

using namespace std;



int main() {

    int age;

    cout << "Enter your age: ";

    cin >> age;  // รับค่าจำนวนเต็มจากผู้ใช้

    cout << "Your age is " << age << endl;



    return 0;

}
```

ผลลัพธ์:

```
Enter your age: 25

Your age is 25
```

ความแตกต่างระหว่าง Input/Output ใน C และ C++

ภาษา	Output	Input
C	printf()	scanf()
C++	cout <<	cin >>

การใช้งานร่วมกับหลายตัวแปร

ภาษา C

- ใช้ , เพื่อเชื่อม Placeholder ใน printf() และหลายตัวแปรใน scanf()

```
#include <stdio.h>



int main() {

    int age;

    float height;



    printf("Enter your age and height: ");

    scanf("%d %f", &age, &height);  // รับค่าหลายตัว

    printf("Age: %d, Height: %.2f\n", age, height);



    return 0;

}
```

ผลลัพธ์:

```
Enter your age and height: 25 5.8

Age: 25, Height: 5.80
```

ภาษา CC++

- ใช้ << และ >> หลายครั้งเพื่อจัดการหลายตัวแปร

```
#include <iostream>

using namespace std;



int main() {

    int age;

    float height;



    cout << "Enter your age and height: ";

    cin >> age >> height;  // รับค่าหลายตัว

    cout << "Age: " << age << ", Height: " << height << endl;



    return 0;

}
```

ผลลัพธ์:

```
Enter your age and height: 25 5.8

Age: 25, Height: 5.8
```

สรุป

1. **ภาษา C:**
    - แสดงผล: printf()
    - รับค่า: scanf()
    - ต้องกำหนดรูปแบบข้อมูล (%d, %f, %c) และใช้ & สำหรับตัวแปร
2. **ภาษา C++:**
    - แสดงผล: cout <<
    - รับค่า: cin >>
    - อ่านง่ายกว่า C และไม่ต้องใช้รูปแบบหรือ & สำหรับตัวแปร

การเลือกใช้งานขึ้นอยู่กับภาษาโปรแกรมที่เลือกพัฒนา โดย C++ สะดวกและกระชับกว่าสำหรับ Input/Output!

1.1.3.แนะนำฟังก์ชัน (Function) และการเรียกใช้ใน C และ C++

ฟังก์ชัน เป็นกลุ่มคำสั่งที่ทำหน้าที่เฉพาะเจาะจงและสามารถนำกลับมาใช้ซ้ำได้ ฟังก์ชันช่วยลดความซ้ำซ้อนและเพิ่มความชัดเจนในโค้ด โดยทั้งภาษา **C** และ **C++** รองรับการสร้างและใช้งานฟังก์ชันอย่างมีประสิทธิภาพ

1. ฟังก์ชันในภาษา C

ส่วนประกอบของฟังก์ชัน

ส่วนการประกาศ (Function Declaration/Prototype): บอกให้โปรแกรมทราบว่ามีฟังก์ชันนี้ในโค้ด เช่น:

```
int add(int a, int b);
```

ส่วนการนิยาม (Function Definition): เป็นการเขียนคำสั่งในฟังก์ชัน เช่น

```
int add(int a, int b) {
    return a + b;
}
```

ส่วนการเรียกใช้ (Function Call): เรียกฟังก์ชันในโค้ด เช่น:

```
int result = add(5, 10);
```

ตัวอย่างโค้ดภาษา C: ฟังก์ชันคำนวณผลบวก

```
#include <stdio.h>


// ฟังก์ชันประกาศ (Prototype)
int add(int a, int b);


int main() {
    int num1 = 10, num2 = 20;


    // เรียกใช้ฟังก์ชัน
    int sum = add(num1, num2);


    printf("Sum: %d\n", sum);  // แสดงผลลัพธ์
    return 0;
}


// ฟังก์ชันนิยาม (Definition)
int add(int a, int b) {
    return a + b;
}
```

ผลลัพธ์:

```
Sum: 30
```

ฟังก์ชันในภาษา C++

ส่วนประกอบเหมือนภาษา C

- ฟังก์ชันใน C++ มีโครงสร้างคล้ายภาษา C แต่มีความสามารถเพิ่มเติม เช่น การรองรับ **Function Overloading**

ตัวอย่างโค้ดภาษา C++: ฟังก์ชันคำนวณผลบวก

```
#include <iostream>

using namespace std;

// ฟังก์ชันประกาศ (Prototype)

int add(int a, int b);

int main() {

    int num1 = 10, num2 = 20;

    // เรียกใช้ฟังก์ชัน

    int sum = add(num1, num2);

    cout << "Sum: " << sum << endl;  // แสดงผลลัพธ์

    return 0;

}

// ฟังก์ชันนิยาม (Definition)

int add(int a, int b) {

    return a + b;

}
```

ผลลัพธ์:

```
Sum: 30
```

3. การใช้งาน Function Overloading (เฉพาะใน C++)

Function Overloading คือการสร้างฟังก์ชันหลายตัวที่มีชื่อเดียวกัน แต่มีจำนวนหรือชนิดของพารามิเตอร์ต่างกัน

ตัวอย่างโค้ด: Function Overloading

```
#include <iostream>

using namespace std;

// Overloading ฟังก์ชัน add

int add(int a, int b) {

    return a + b;

}

float add(float a, float b) {

    return a + b;

}

int main() {

    cout << "Sum (int): " << add(5, 10) << endl;        // เรียก add(int, int)

    cout << "Sum (float): " << add(3.5f, 4.5f) << endl; // เรียก add(float, float)



    return 0;

}
```

ผลลัพธ์:

```
Sum (int): 15

Sum (float): 8
```

4. ฟังก์ชันแบบไม่มีการคืนค่า (Void Function)

ฟังก์ชันที่ไม่คืนค่าใด ๆ จะใช้ชนิด void

ตัวอย่างโค้ด: Void Function

```
#include <iostream>

using namespace std;

void printMessage() {

    cout << "Hello, this is a void function!" << endl;

}

int main() {

    printMessage();  // เรียกฟังก์ชัน void

    return 0;

}
```

ผลลัพธ์:

```
Hello, this is a void function!
```

5. ฟังก์ชันแบบส่งค่าผ่านพอยน์เตอร์ (ภาษา C)

ภาษา C รองรับการส่งค่าผ่าน **พอยน์เตอร์** เพื่อให้ฟังก์ชันเปลี่ยนแปลงค่าของตัวแปรต้นฉบับได้

ตัวอย่างโค้ด: ส่งค่าผ่านพอยน์เตอร์

```
#include <stdio.h>


void increment(int *num) {
    *num = *num + 1;  // เพิ่มค่าผ่านพอยน์เตอร์
}


int main() {
    int value = 10;


    printf("Before: %d\n", value);
    increment(&value);  // ส่งที่อยู่ของตัวแปร
    printf("After: %d\n", value);


    return 0;
}
```

ผลลัพธ์:

```
Before: 10
After: 11
```

6. การเรียกใช้ฟังก์ชันโดยอ้างอิง (C++)

ภาษา C++ สามารถส่งค่าผ่าน **Reference** โดยไม่ต้องใช้พอยน์เตอร์

ตัวอย่างโค้ด: Pass by Reference

```
#include <iostream>

using namespace std;



void increment(int &num) {

    num = num + 1;  // เพิ่มค่าผ่าน Reference

}



int main() {

    int value = 10;



    cout << "Before: " << value << endl;

    increment(value);  // ส่งตัวแปรโดยอ้างอิง

    cout << "After: " << value << endl;

    return 0;

}
```

ผลลัพธ์:

```
Before: 10

After: 11
```

7. สรุป

ลักษณะการใช้งาน	ภาษา C	ภาษา C++
ฟังก์ชันทั่วไป	ใช้ Function Declaration และ Definition	ใช้ Function Declaration และ Definition
Function Overloading	ไม่รองรับ	รองรับ
Pass by Pointer	ใช้ * และ &	ใช้ * และ &
Pass by Reference	ไม่รองรับ	รองรับ &
Void Function	ใช้ void	ใช้ void

การใช้งานฟังก์ชันช่วยให้โค้ดเป็นระเบียบ ลดความซ้ำซ้อน และเพิ่มความยืดหยุ่นในการพัฒนาโปรแกรม

1.2. ปฏิบัติ (3 ชม.)

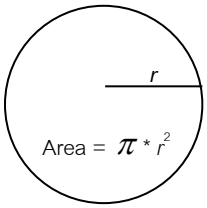
รหัสนักศึกษา_____

ชื่อนักศึกษา_____

ตัวอย่างที่ 1 จงวิเคราะห์งานสำหรับการเขียนโปรแกรมคำนวณหาพื้นที่วงกลม

1) วัตถุประสงค์ของการเขียนโปรแกรม – เพื่อคำนวณหาพื้นที่วงกลม

2) รูปแบบผลลัพธ์ที่ต้องการ



- แสดงผลออกทางจอภาพดังนี้

+++ Results +++

Radius = 7

Area = 154

หรือ - 2.1) แสดงรัศมีของรูปวงกลมที่ต้องการหา

2.2) แสดงพื้นที่ของรูปวงกลมที่คำนวณได้

3) ข้อมูลนำเข้า คือ รัศมีของวงกลม

4) ตัวแปรที่ใช้

R = ตัวแปรสำหรับเก็บค่ารัศมีวงกลม

Area = ตัวแปรสำหรับเก็บพื้นที่ซึ่งเป็นผลลัพธ์ที่ได้จากสูตร Area = π*(R^2)

5) วิธีการประมวลผล

5.1) เริ่มต้นทำงาน

5.2) รับค่าตัวแปร R

5.3) คำนวณพื้นที่วงกลม Area = π*(R^2)

5.4) พิมพ์ค่าตัวแปร R และพิมพ์ค่าผลลัพธ์จากตัวแปร Area

5.5) จบการทำงาน

ตัวอย่างที่2 จงวิเคราะห์งานสำหรับการเขียนโปรแกรมตรวจสอบผลการสอบนักศึกษา

โปรแกรมตรวจสอบผลการสอบนักศึกษา

- หากคะแนนน้อยกว่า 50 คะแนน ให้แสดงข้อความว่า **“Failed!”** ออกทางหน้าจอ
- หากคะแนนมากกว่าหรือเท่ากับ 50 คะแนน ให้แสดงข้อความว่า **“Pass!”** ออกทางหน้าจอ

1) วัตถุประสงค์ของการเขียนโปรแกรม – เพื่อตรวจสอบผลการสอบของนักศึกษา

2) รูปแบบผลลัพธ์ที่ต้องการ

- แสดงผลออกทางจอภาพดังนี้

Score = 55
Pass!

หรือ - 2.1) แสดงคะแนนนักศึกษา

2.2) แสดงผลว่าผ่านหรือไม่ผ่าน

3) ข้อมูลนำเข้า คือ คะแนนของนักศึกษา

4) ตัวแปรที่ใช้

score = ตัวแปรสำหรับเก็บคะแนนนักศึกษา

5) วิธีการประมวลผล

5.1) เริ่มต้นทำงาน

5.2) รับค่าตัวแปร score

5.3) ตรวจสอบว่าคะแนนน้อยกว่า 50 หรือไม่

5.4) พิมพ์ค่าตัวแปร score และผลจากการตรวจสอบโดยที่

5.4.1) ถ้าเป็นจริง : แสดงข้อความว่า “Failed!”

5.4.2) ถ้าเป็นเท็จ : แสดงข้อความว่า “Pass!”

5.5) จบการทำงาน

ตัวอย่างที่ 3 จงเขียนผังงานโปรแกรมที่ได้จากการวิเคราะห์งานในตัวอย่างที่ 1

จากการวิเคราะห์งานในตัวอย่างที่ 1 จะได้ว่าขั้นตอนวิธีการประมวลผลของโปรแกรม มีดังนี้

1) เริ่มต้นทำงาน

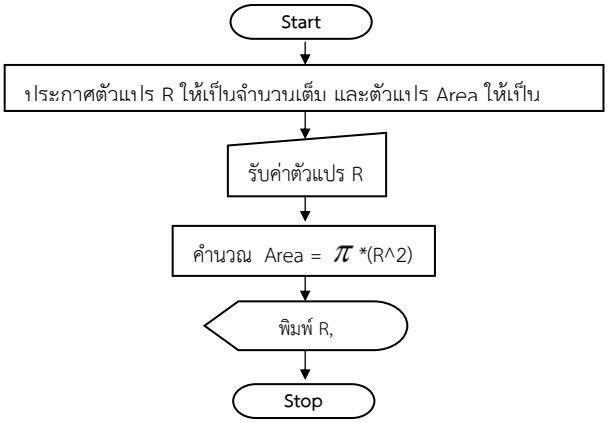
2) รับค่าตัวแปร R

3) คำนวณพื้นที่วงกลม Area = *(R^2)

4) พิมพ์ค่าตัวแปร R และพิมพ์ค่าผลลัพธ์จากตัวแปร Area

5) จบการทำงาน

การเขียนผังงานโปรแกรมจะนำขั้นตอนวิธีการประมวลผลของโปรแกรมที่ได้จากการวิเคราะห์งาน สามารถเขียนผังงานได้ดังนี้



ตัวอย่างที่ 4 จงเขียนผังงานโปรแกรมที่ได้จากการวิเคราะห์งานในตัวอย่างที่ 2.32

จากการวิเคราะห์งานในตัวอย่างที่ 3 จะได้ว่าขั้นตอนวิธีการประมวลผลของโปรแกรม มีดังนี้

1) เริ่มต้นทำงาน

2) รับค่าตัวแปร score

3) ตรวจสอบว่าคะแนนน้อยกว่า 50 หรือไม่

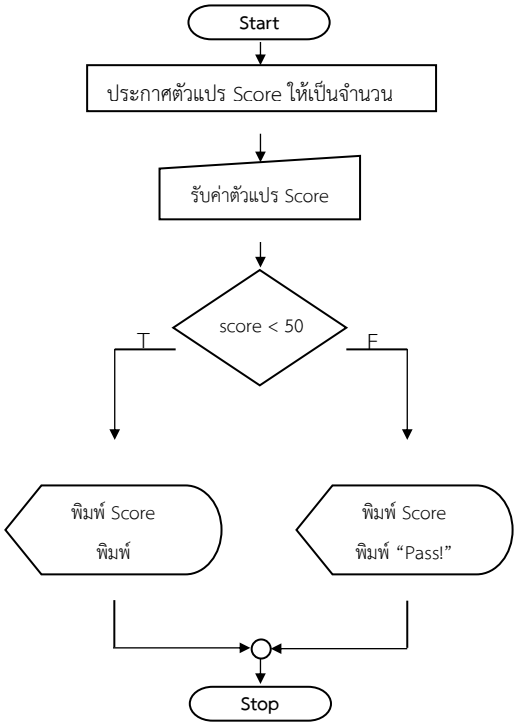
4) พิมพ์ค่าตัวแปร score และผลจากการตรวจสอบโดยที่

4.1) ถ้าเป็นจริง : แสดงข้อความว่า “Failed!”

4.2) ถ้าเป็นเท็จ : แสดงข้อความว่า “Pass!”

5) จบการทำงาน

การเขียนผังงานโปรแกรมจะนำขั้นตอนวิธีการประมวลผลของโปรแกรมที่ได้จากการวิเคราะห์งาน สามารถเขียนผังงานได้ดังนี้



1.2.1.ให้นักศึกษาเขียนวิเคราะห์งานของการเขียนโปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้า

1) วัตถุประสงค์ของการเขียนโปรแกรม

.....

.....

.....

2) รูปแบบผลลัพธ์ที่ต้องการ

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

3) ข้อมูลนำเข้า

.....

.....

.....

4) ตัวแปรที่ใช้

.....

.....

.....

.....

.....

.....

5) วิธีการประมวลผล

.....

.....

.....

.....

.....

.....

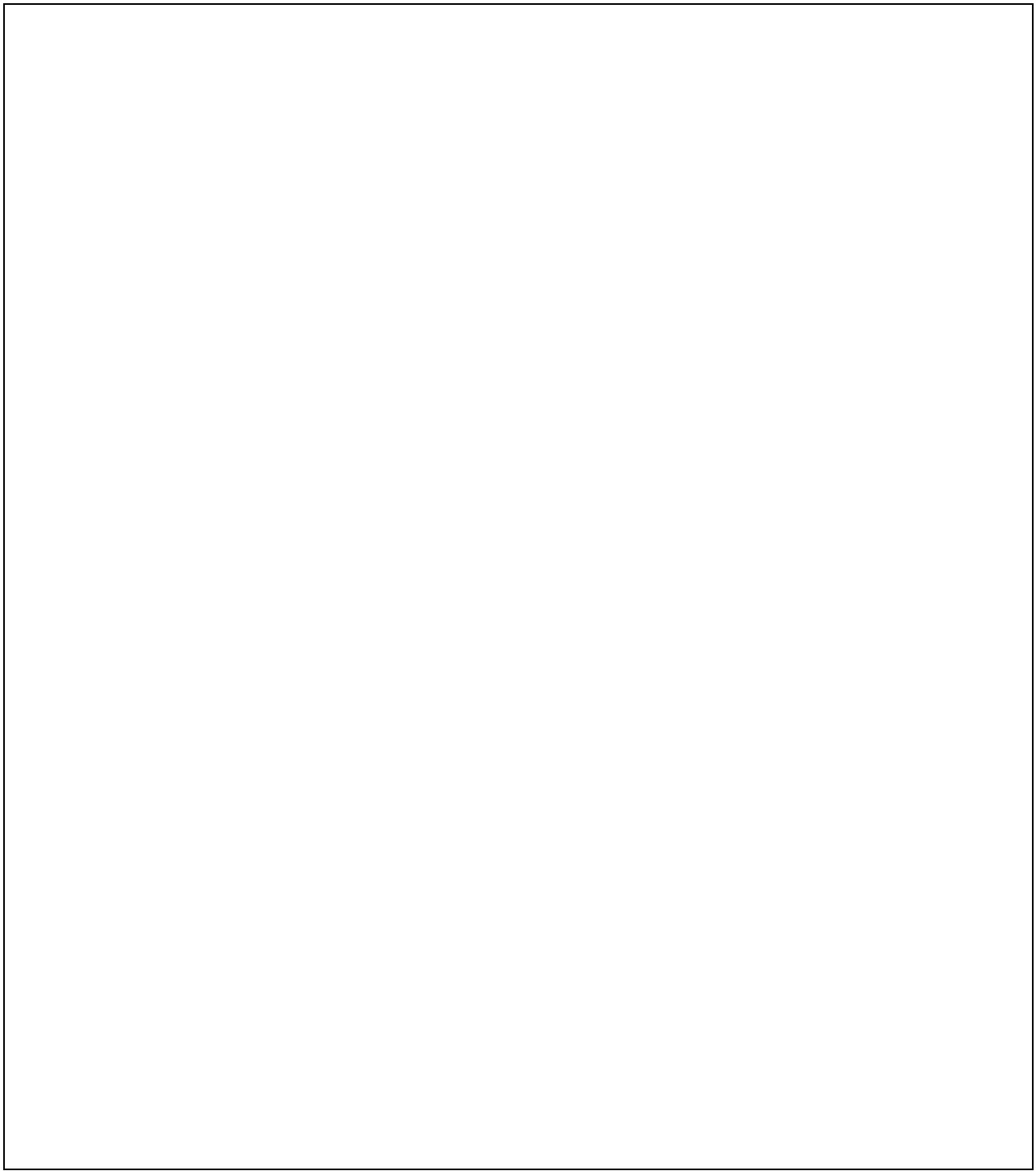
.....

.....

.....

.....

ให้นักศึกษาเขียนผังงานโปรแกรม ที่ได้จากการการวิเคราะห์งานของการเขียนโปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้าในข้อ 2.2.1



1.2.2.สร้างฟังก์ชันสำหรับคำนวณปริมาณของรูปทรงต่างๆ

นักศึกษาเลือกรูปทรง

.....

1) วัตถุประสงค์ของการเขียนโปรแกรม

.....

.....

.....

2) รูปแบบผลลัพธ์ที่ต้องการ

.....

.....

.....

.....

.....

.....

.....

.....

.....

3) ข้อมูลนำเข้า

.....

.....

.....

4) ตัวแปรที่ใช้

.....

.....

.....

.....

.....

.....

5) วิธีการประมวลผล

.....

.....

.....

.....

.....

.....

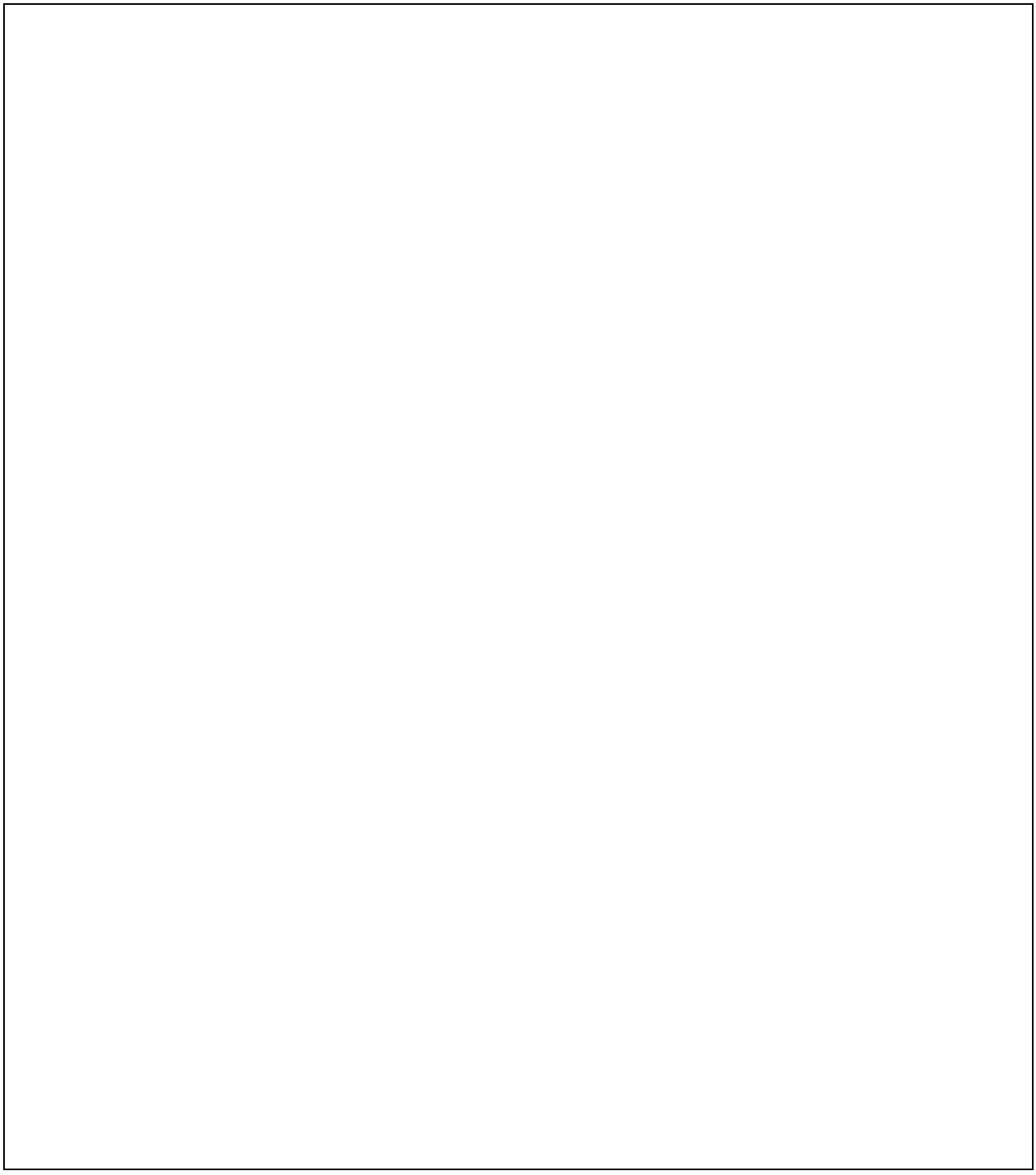
.....

.....

.....

.....

ให้นักศึกษาเขียนผังงานโปรแกรม ที่ได้จากการการวิเคราะห์งานของการเขียนโปรแกรมคำนวณพื้นที่สี่เหลี่ยมผืนผ้าในข้อ 2.2.1



1.2.3.ฝึกเขียนโปรแกรมเพื่อทำความเข้าใจกับการทำงานของฟังก์ชัน

ให้นักศึกษาเขียนการวิเคราะห์งานของโปรแกรมคำนวณเงินเดือนของพนักงานดังต่อไปนี้

โปรแกรมคำนวณเงินเดือนของพนักงาน

- อ่านค่ารหัสพนักงาน ชั่วโมงการทำงาน และอัตราค่าแรง
- ตรวจสอบชั่วโมงการทำงานของพนักงานโดยแบ่งการคิดค่าจ้างออกเป็น 2 กรณี คือ

  กรณีทำงานน้อยกว่า 30 ชั่วโมง : ค่าจ้าง = ชั่วโมงการทำงาน * อัตราค่าแรง

  กรณีทำงาน 30 ชั่วโมง ขึ้นไป : ค่าจ้าง = ชั่วโมงการทำงาน * (อัตราค่าแรง + 50)
- พนักงานทุกคนต้องจ่ายภาษี 7% ของค่าจ้าง
- พิมพ์ข้อมูล รหัสพนักงาน ค่าจ้างที่ยังไม่ถูกหักภาษี จำนวนภาษีที่ต้องจ่าย และค่าจ้างสุทธิที่ได้รับจริง

  ลงบนกระดาษเพื่อส่งให้พนักงานแต่ละคน

1) วัตถุประสงค์ของการเขียนโปรแกรม

.....

.....

.....

2) รูปแบบผลลัพธ์ที่ต้องการ

.....

.....

.....

.....

3) ข้อมูลนำเข้า

.....

.....

.....

4) ตัวแปรที่ใช้

.....

.....

.....

.....

.....

.....

5) วิธีการประมวลผล

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

ให้นักศึกษาเขียนผังงานโปรแกรม (Flowchart) ที่ได้จากการการวิเคราะห์งานของการเขียนโปรแกรมข้อ 2.2.3

โครงสร้างของภาษา C/C++

ภาษา C และ C++ มีโครงสร้างพื้นฐานที่คล้ายคลึงกัน โดยสามารถแบ่งออกเป็น 6 ส่วนหลัก ดังนี้:

1. การนำเข้าหัวข้อไลบรารี: ใช้สำหรับเรียกใช้ฟังก์ชันหรือคลาสสำเร็จรูป
2. การประกาศข้อมูล: กำหนดตัวแปรหรือค่าที่ใช้ทั้งโปรแกรม
3. ฟังก์ชันหลัก: จุดเริ่มต้นการทำงานของโปรแกรม
4. ฟังก์ชันอื่น: สำหรับการแบ่งงานหรือสร้างฟังก์ชันเฉพาะ
5. ตัวแปรและนิพจน์: ใช้สำหรับเก็บและประมวลผลข้อมูล
6. คำอธิบาย: ช่วยให้โค้ดเข้าใจง่ายขึ้น

ให้นักศึกษาเขียนลงในช่องว่าง A-I ว่าแต่ละส่วนเรียกว่าอะไร

```
#include <iostream>  // A)_____________________________
int globalVar = 10;  // B)_____________________________

// C_____________________________
int addNumbers(int a, int b) {
    return a + b;  // D)_____________________________
}
// E)_____________________________ _____________________________
int main() {
    // E)_____________________________________________
    int num1 = 20, num2 = 30;

    // G)_____________________________________________________
    int sum = addNumbers(num1, num2);

    // H)_____________________________________________________
    std::cout << "Sum: " << sum << std::endl;

    return 0;  // I)_____________________________________
}
```